edv-buchversand.de

Numerisches Python

Arbeiten mit NumPy, Matplotlib und Pandas

» Hier geht's direkt zum Buch

DIE LESEPROBE

Diese Leseprobe wird Ihnen von www.edv-buchversand.de zur Verfügung gestellt.

31 Finanzverwaltung mit Pandas

In diesem Kapitel führen wir nichts Neues ein, sondern wollen das Erlernte an zwei interessanten Fallbeispielen einüben. Wir haben das Kapitel etwas hochtrabend „Finanzverwaltung“ genannt. In unseren Beispielen geht es jedoch nur um die Implementierung zweier einfacher, aber dennoch überaus wichtiger Konzepte:

- Haushaltsbuch
- Einnahmeüberschussrechnung

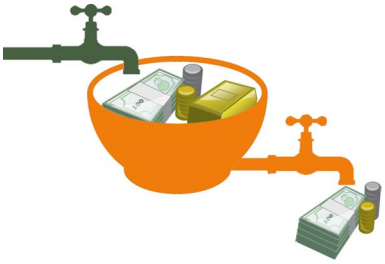


Bild 31.1 Geldfluss

In jedem dieser Beispiele geht es um die Verwaltung von Geldern: im ersten Fall für den privaten Bereich und im zweiten für den kommerziellen Bereich, also mit Mehrwertsteuer. Gemeinsam ist beiden Beispielen auch, dass wir die Daten zur Verarbeitung aus einer Datei beziehen, die wir mit Pandas einlesen.

31.1 Haushaltsbuch

Zuerst geht es nun um einfache Ausgaben- und Einkommenstabellen für den privaten Gebrauch. Einige sagen, wenn man Geld erfolgreich verwalten möchte, muss man die Einnahmen und Ausgaben genau verfolgen. Die Überwachung des Geldflusses ist wichtig, um die eigene finanzielle Situation besser zu verstehen. Man muss genau wissen, wie viel ein- und ausgeht. Dazu bedient man sich üblicherweise eines Haushaltsbuches, in dem alle Ein- und Ausgaben protokolliert sind. Dieses Kapitel will niemanden von der Notwendigkeit überzeugen, dies zu tun. Das Hauptaugenmerk liegt vielmehr auf den Möglichkeiten, die Python und Pandas bieten, um die erforderlichen Tools zu programmieren. Wir zeigen Verfahren, um ein Haushaltsbuch auszuwerten.

31.1.1 Haushaltsbuch mit CSV-Datei

Wir beginnen mit einem kleinen Beispiel, das für private Zwecke geeignet ist. Im folgenden Kapitel unseres Pandas-Tutorials wird ein ausführlicheres Beispiel behandelt, welches für kleine Unternehmen geeignet ist.

```
import pandas as pd
ein_aus_df = pd.read_csv("ein_und_ausgaben.csv", sep=";")
print(ein_aus_df[:12])  # Ausgabe der ersten zwölf Zeilen
```

Ausgabe:

```
         Datum          Beschreibung               Kategorie  Ausgaben
         ↪  Einnahmen
0   2022-07-01          Gehalt Frank               Einkommen      0.00
↪  4896.44
1   2022-07-02    Numerisches Python         Kultur, Bildung     29.00
↪  0.00
2   2022-07-04            Supermarkt      Essen und Getränke    132.40
↪  0.00
3   2022-07-04                Miete                   Miete   1267.00
↪  0.00
4   2022-07-04          Gehalt Laura               Einkommen      0.00
↪  4910.14
5   2022-07-04                Strom          Betriebskosten     87.34
↪  0.00
6   2022-07-08   Wasser und Abwasser          Betriebskosten     60.56
↪  0.00
7   2022-07-10  Fitnessstudio, Sarah    Gesundheit und Sport     19.00
↪  0.00
8   2022-07-11                 Bank           Kredittilgung   1287.43
↪  0.00
9   2022-07-12  LeGourmet Restaurant  Restaurants und Hotels    145.00
↪  0.00
10  2022-07-13            Supermarkt      Essen und Getränke    197.42
↪  0.00
11  2022-07-13   Pizzeria Pulcinella  Restaurants und Hotels     60.00
↪  0.00
```

Durch Lesen der CSV-Datei haben wir ein DataFrame-Objekt erstellt. Was können wir damit machen oder mit anderen Worten: Welche Informationen interessieren Frank und Sarah? Natürlich interessieren sie sich für den Kontostand. Sie wollen wissen, wie hoch das Gesamteinkommen war, und sie wollen die Summe aller Ausgaben sehen. Die Salden ihrer Ausgaben und Einnahmen können leicht berechnet werden, indem die Funktion `sum` auf das DataFrame-Objekt `ein_aus_df[['Ausgaben', 'Einnahmen']]` angewendet wird:

```
print(ein_aus_df[['Ausgaben', 'Einnahmen']].sum())
```

Ausgabe:

```
Ausgaben      7660.44
Einnahmen    19613.16
dtype: float64
```

Welche weiteren Informationen möchten Sie aus den Daten gewinnen? Sie könnten daran interessiert sein, die Ausgaben nach den verschiedenen Kategorien zusammengefasst zu sehen. Dies lässt sich mittels `groupby` und `sum` bewerkstelligen:

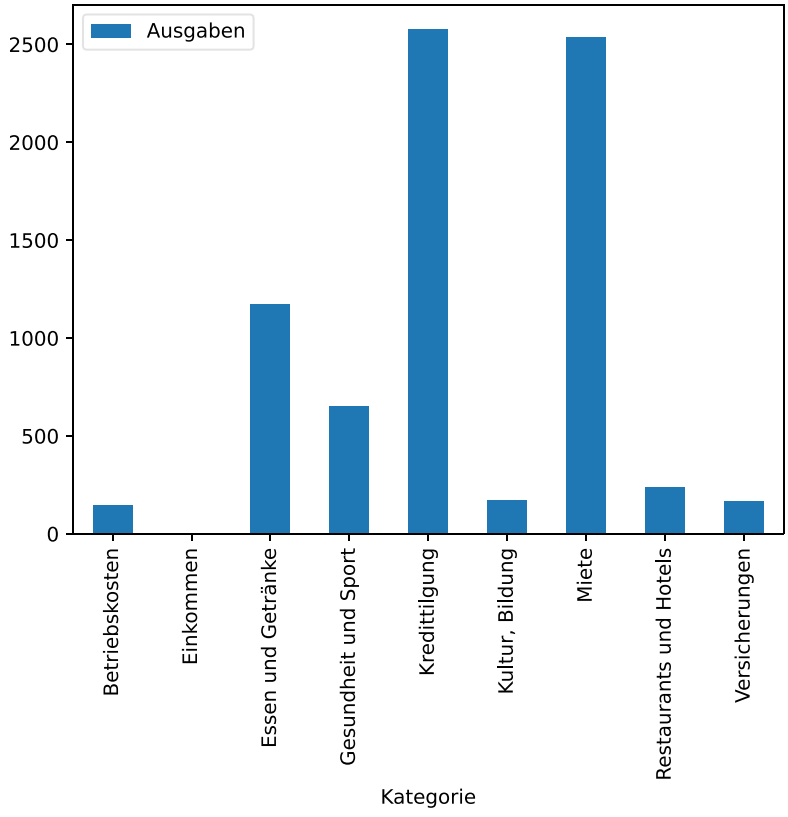
```
category_sums = ein_aus_df.groupby("Kategorie").sum()
print(category_sums)
```

Ausgabe:

```
                      Ausgaben  Einnahmen
Kategorie
Betriebskosten          147.90       0.00
Einkommen                 0.00   19613.16
Essen und Getränke     1174.61       0.00
Gesundheit und Sport    650.18       0.00
Kredittilgung          2574.86       0.00
Kultur, Bildung         173.00       0.00
Miete                  2534.00       0.00
Restaurants und Hotels  238.00       0.00
Versicherungen          167.89       0.00
```

Man kann sich die Ein- und Ausgaben auch als Balkendiagramme ausgeben lassen:

```
ax = category_sums.plot.bar(y="Ausgaben", rot=90)
```



Am besten eignen sich in diesem Falle Tortendiagramme:

```
ax = category_sums.plot.pie(y="Ausgaben")
ax.legend(loc="upper left", bbox_to_anchor=(1.5, 1))
```

31.1.2 Erzeugen eines Excel-Haushaltsbuches

Wenn man sich vorstellt, dass man die ganze Zeit Kategorienamen wie „Essen und Getränke“ oder „Restaurants und Hotels“ eingeben muss, kann man sich leicht vorstellen, dass dies zum einen sehr umständlich und zum anderen sehr fehleranfällig ist, da sich leicht Tippfehler im Ausgaben- und Einnahmenjournal einschleichen können.

Es ist daher eine gute Idee, Nummern (Kontonummern) für die Kategorien zu verwenden. Gleichzeitig möchte man aber auch die Kategorien in textueller Form erhalten. Wir wollen nun aus den vorigen Daten eine Excel-Datei erzeugen, die zwei Tabellen (engl. sheets) besitzt. Die erste Tabelle wird die Zuordnung der Kategorien und der Kontonummern enthalten. Die folgenden Kategorien mit Kontonummern sind in unserem Beispiel verfügbar:

Kategorie	Kontonummer
Kredittilgung	200
Versicherungen	201
Essen und Getränke	202
Kultur, Bildung	203
Transport	204
Gesundheit und Sport	205
Haushalt und Dienstleistungen	206
Kleidung	207
Kommunikationskosten	208
Restaurants und Hotels	209
Betriebskosten	210
andere Ausgaben	211
Einkommen	400

Wir können dies als Python-Dictionary implementieren, das Kategorien in Kontonummern abbildet:

```
category2account = {"Kredittilgung": "200",
                    "Versicherungen": "201",
                    "Essen und Getränke": "202",
                    "Kultur, Bildung": "203",
                    "Transport": "204",
                    "Gesundheit und Sport": "205",
                    "Haushalt und Dienstleistungen": "206",
                    "Kleidung": "207",
                    "Kommunikationskosten": "208",
                    "Restaurants und Hotels": "209",
                    "Betriebskosten": "210",
                    "Miete": "211",
                    "andere Ausgaben": "220",
                    "Einkommen": "400"}
```

Im nächsten Schritt werden wir eine Spalte mit den Kontonummern in unser DataFrame-Objekt `ein_aus_df` einfügen. Dazu formulieren wir zuerst eine Funktion `category2account_func`, die Kategorien in Kontonummern wandelt. Diese Funktion benutzen wir dann in der Methode `apply`, angewendet auf die Spalte `Kategorie`:

```
def category2account_func(category):
    if category in category2account:
        return category2account[category]
    else:
        # doesn't exist
        return '000'


acc = ein_aus_df['Kategorie'].apply(category2account_func)
# Anhängen der 'Konto'-Spalte ans Ende:
#ein_aus_df['Konto'] = acc

# Einfügen als Spalte mit Index 2:
ein_aus_df.insert(loc=2,
                  column='Konto',
                  value=acc)
# Löschen der 'Kategorie'-Spalte:
ein_aus_df.drop('Kategorie', axis=1, inplace=True)
print(ein_aus_df[:7])
```

Ausgabe:

```
         Datum         Beschreibung Konto  Ausgaben  Einnahmen
0  2022-07-01         Gehalt Frank   400      0.00    4896.44
1  2022-07-02   Numerisches Python   203     29.00       0.00
2  2022-07-04          Supermarkt   202    132.40       0.00
3  2022-07-04               Miete   211   1267.00       0.00
4  2022-07-04         Gehalt Laura   400      0.00    4910.14
5  2022-07-04               Strom   210     87.34       0.00
6  2022-07-08  Wasser und Abwasser   210     60.56       0.00
```

Wir werden dieses DataFrame-Objekt ebenfalls in unserer Excel-Datei als eine separate Tabelle abspeichern. Diese Excel-Datei wird somit zwei Tabellen enthalten: eine mit dem Namen „Journal“ und die andere mit der Zuordnung von Konto zu Kategoriennamen unter dem Namen „Kontonummern“.

```
account_numbers = pd.Series(category2account)
account_numbers.name = "Kontonummern"

with pd.ExcelWriter('ein_und_ausgaben.xlsx') as writer:
    account_numbers.to_excel(writer, "Kontonummern")
    ein_aus_df.to_excel(writer, "Journal")
```

31.1.3 Auswertung des Excel-Haushaltsbuches

Jetzt kann man diese Excel-Datei als Grundlage für das Haushaltsbuch nehmen. Wir müssen unser obiges Programm nun entsprechend anpassen. Zunächst lesen wir die Excel-Datei ein:

```
import pandas as pd

with pd.ExcelFile('ein_und_ausgaben.xlsx') as xl:
    print(xl.sheet_names)
    accounts = xl.parse('Kontonummern', index_col=1)
    journal = xl.parse('Journal', index_col=0)

kat = journal.Konto.replace(accounts.index,
                            accounts.values)
journal['Kategorie'] = kat
print(accounts[:5])
print(journal[:5])
```

Ausgabe:

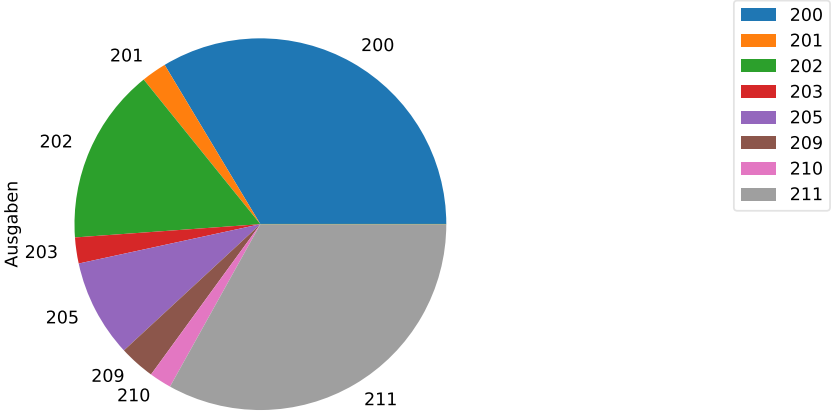
```
                     Unnamed: 0
Kontonummern
200                Kredittilgung
201               Versicherungen
202           Essen und Getränke
203              Kultur, Bildung
204                    Transport
        Datum        Beschreibung  Konto  Ausgaben  Einnahmen
        ↪  Kategorie
0  2022-07-01        Gehalt Frank    400       0.0    4896.44
↪  Einkommen
1  2022-07-02  Numerisches Python    203      29.0       0.00     Kultur,
↪  Bildung
2  2022-07-04          Supermarkt    202     132.4       0.00  Essen und
↪  Getränke
3  2022-07-04               Miete    211    1267.0       0.00
↪  Miete
4  2022-07-04        Gehalt Laura    400       0.0    4910.14
↪  Einkommen
```

Wir erzeugen nun ein Tortendiagramm aus den Daten nach den Kategorien gruppiert:

```
category_sums = journal.groupby('Kategorie').sum()
ax = category_sums.plot.pie(y="Ausgaben")
ax.legend(loc="upper left", bbox_to_anchor=(1.5, 1))
```

Nach Kontonummern können wir ähnlich leicht gruppieren:

```
konten_summen = journal.groupby('Konto').sum()
ax = konten_summen.plot.pie(y="Ausgaben")
ax.legend(loc="upper left", bbox_to_anchor=(1.5, 1))
```



31.2 Einnahmenüberschussrechnung

Bei der in Deutschland benutzten Einnahmenüberschussrechnung (EÜR), in Österreich als Einnahmen-Ausgaben-Rechnung (E/A-Rechnung) bekannt, handelt es sich um eine vereinfachte Gewinnermittlungsmethode, die so vom Gesetz vorgegeben und für bestimmte

Berufsgruppen anerkannt ist. Die Einnahmenüberschussrechnung ist sowohl in Deutschland als auch in Österreich im § 4 Abs. 3 des jeweiligen Einkommensteuergesetzes (EStG) geregelt. Alle Steuerpflichtigen, die nicht zur doppelten Buchführung verpflichtet sind, dürfen dieses vereinfachte Verfahren zur Gewinnermittlung verwenden.

Seit Anfang 2013 wurde sie auch in der Schweiz mit dem neuen Rechnungslegungsrecht eingeführt. Seitdem sind kleine Unternehmen auch in der Schweiz nicht mehr dazu verpflichtet, ihre Bücher doppelt zu führen. Stattdessen reicht nun eine einfache Buchhaltung, auch Milchbüechli-Rechnung genannt, was der deutschen EÜR entspricht.

Bei der EÜR gilt das Zufluss- und Abflussprinzip, was bedeutet, dass lediglich die Einnahmen bzw. Ausgaben zu berücksichtigen sind, die in dem entsprechenden Wirtschaftsjahr vereinnahmt bzw. gezahlt wurden. Bestandsveränderungen bleiben unberücksichtigt. Auch wenn hier und an anderen Stellen immer von „vereinfachter“ oder von „einfacher“ Gewinnermittlungsmethode die Rede ist, so darf das nicht darüber hinwegtäuschen, dass auch bei dieser Methode viele gesetzlichen Regeln zu beachten sind. Im Rahmen dieses Buches kann natürlich nicht auf die komplexe Materie eingegangen werden. Die hier vorgestellten Verfahren können für die eigene EÜR verwendet werden, aber es kann keine Garantie auf die Richtigkeit gegeben werden.

Wir sind hauptsächlich daran interessiert, die verschiedenen Möglichkeiten vorzustellen, wie Pandas und Python zur Einnahmeüberschussrechnung benutzt werden können. Wir zeigen, wie es mit Python möglich ist, den Geldfluss zu überwachen und zu visualisieren. Auf diese Weise kann man sich einen besseren Überblick über die finanzielle Situation des Betriebes oder der selbständigen Tätigkeit verschaffen. Die hier vorgestellten Algorithmen können auch steuerlich eingesetzt werden. Aber seien Sie gewarnt: Dies ist eine sehr allgemeine Behandlung der Angelegenheit und muss an die tatsächliche Steuersituation angepasst werden, d. h. es kann keine Gewähr für die steuerliche Richtigkeit übernommen werden.

31.2.1 Journaldatei

Als Grundlage für unsere Einnahmeüberschussrechnung dient ein Excel-Dokument mit zwei Tabellen: eine mit den laufenden Posten, also den Einnahmen und Ausgaben in chronologischer Reihenfolge. Diese Tabelle bezeichnen wir als Journal. Die zweite Tabelle enthält die Zuordnungen von Kontonummern zu den Kontobezeichnungen. Im Folgenden lesen wir diese Excel-Datei in zwei DataFrame-Objekte ein:

```
import pandas as pd

with pd.ExcelFile("einnahme_ueber_2022.xlsx") as xl:
    konto_beschreibung = xl.parse("kontenplan",
                                  index_col=0)
    journal = xl.parse("journal",
                       index_col=0)
print(f"Die ersten 15 Zeilen von journal:\n{journal[:12]}")
print(f"Kontobeschreibung:\n{konto_beschreibung}")
```

Ausgabe:

```
Die ersten 15 Zeilen von journal:
             Kto  DokuNummer           Beschreibung  StSatz    Brutto
Datum
2022-04-02  4402  8983233038           Zurkan, Köln      19   4105.98
2022-04-02  2010    57550799  Bengelmann, Souvenirs      19  -1890.00
2022-04-02  2200    14989004               Gehälter       0 -17478.23
2022-04-02  2500    12766279     Tankstelle, Benzin      19    -89.40
2022-04-02  4400  3733462359     EnergyCom, Hamburg      19   4663.54
2022-04-02  4402  7526058231     Enoigo, Strasbourg      19   2412.82
2022-04-05  4402  1157284466      Qbooks, Frankfurt       7   2631.42
2022-04-05  4402  7009463592           Qbooks, Köln       7   3628.45
2022-04-05  2020    68433353        Jamdon, Kleider      19  -1900.00
2022-04-05  2010    53353169      Outleg, Souvenirs      19  -2200.00
2022-04-09  4402  4775929332     Drupa AG, Freiburg      19   4751.66
2022-04-09  2100    10759896    Gebäudeversicherung       0   -467.00
Kontobeschreibung:
            Beschreibung
Kto
4400    Standort München
4401  Standort Frankfurt
4402     Standort Berlin
2010           Souvenirs
2020             Kleider
2030      Andere Artikel
2050              Bücher
2100      Versicherungen
2200            Gehälter
2300             Kredite
2400              Hotels
2500              Benzin
2600   Telekommunikation
2610            internet
```

31.2.2 Analyse und Visualisierung der Daten

Es gibt viele Möglichkeiten, diese Daten zu analysieren. Wir können zum Beispiel alle Konten zusammenfassen:

```
konten_summen = journal[["Kto", "Brutto"]].groupby("Kto").sum()
print(konten_summen)
```

Ausgabe:

```
         Brutto
Kto
2010  -4090.00
2020 -10500.80
2030  -1350.00
2050   -900.00
2100   -612.00
2200 -69912.92
2300 -18791.92
```

```
2400  -1597.10
2500    -89.40
2600   -492.48
2610   -561.00
4400  37771.84
4401  69610.35
4402  61593.99
```

Nun wollen wir die Daten mittels Kreisdiagrammen, auch Kuchendiagramme genannt, visualisieren. Im Englischen bezeichnet man sie als „pie charts“. Nun haben wir aber ein kleines Problem: Kreisdiagramme dürfen keine negativen Werte enthalten. Dies lässt sich jedoch schnell beheben. Wir können die Konten in Einnahmen- und Ausgabenkonten aufteilen. Das entspricht natürlich auch mehr dem, was wir wirklich sehen wollen.

Wir erstellen zunächst ein DataFrame mit den Einnahmen und eines mit den Ausgaben. Nach der Erzeugung der Ausgabensummen multiplizieren wir das Ergebnis mit -1, um die Werte positiv zu machen:

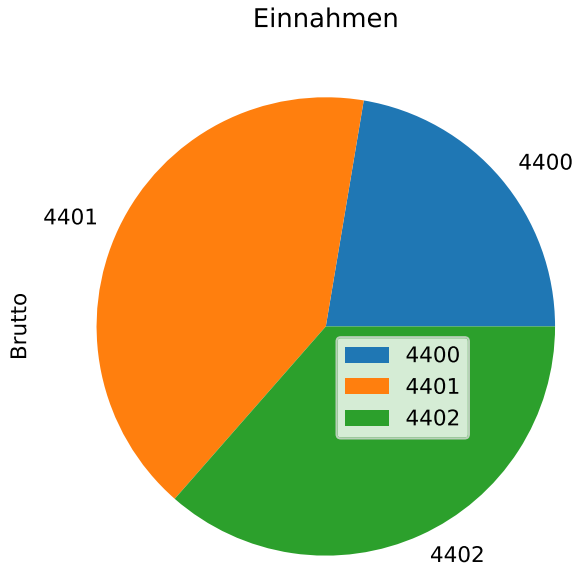
```
einnahmen = konten_summen[konten_summen["Brutto"] > 0]
ausgaben = konten_summen[konten_summen["Brutto"] < 0] * -1

print(f"---- Einnahmen -----\n{einnahmen}")
print(f"---- Ausgaben  -----\n{ausgaben}")
```

Ausgabe:

```
---- Einnahmen -----
        Brutto
Kto
4400  37771.84
4401  69610.35
4402  61593.99
---- Ausgaben  -----
        Brutto
Kto
2010   4090.00
2020  10500.80
2030   1350.00
2050    900.00
2100    612.00
2200  69912.92
2300  18791.92
2400   1597.10
2500     89.40
2600    492.48
2610    561.00
```

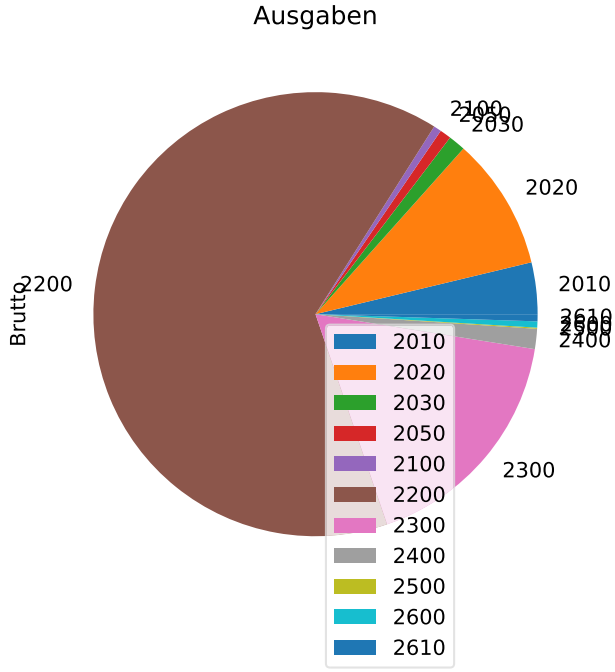
Nun erzeugen wir ein Tortendiagramm mit den Einnahmen:

```
ax = einnahmen.plot(y='Brutto',
                    title='Einnahmen',
                    kind="pie")
ax.legend(bbox_to_anchor=(0.5, 0.5),
          loc="upper left")
```



Nun erzeugen wir analog das entsprechende Tortendiagramm für die Ausgaben.

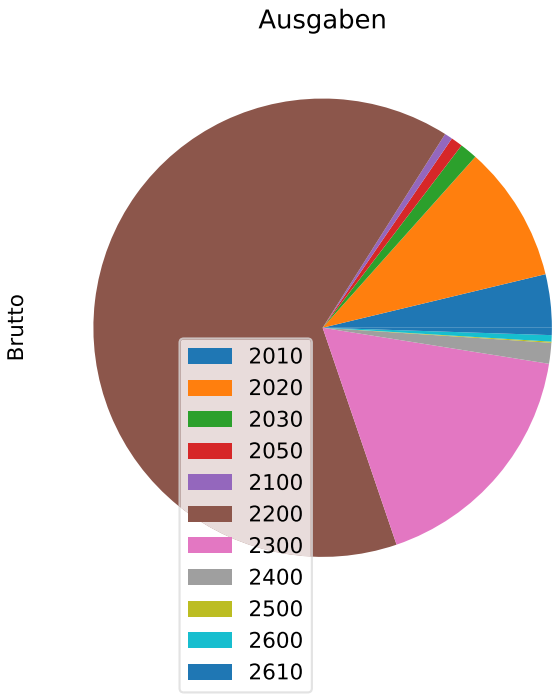
```
ax = ausgaben.plot(y='Brutto',
                   title='Ausgaben',
                   kind="pie")
ax.legend(bbox_to_anchor=(0.5, 0.5),
          loc="upper left")
```



Es gibt zwei Möglichkeiten zu verhindern, dass sich die Labels nicht überlappen. Die erste besteht einfach darin, dass man sie nicht ausgibt. Man hat ja immer noch die nötige Information durch die Farbzuordnung in der Legende.

Nun erzeugen wir analog die Labels nur in der Legende:

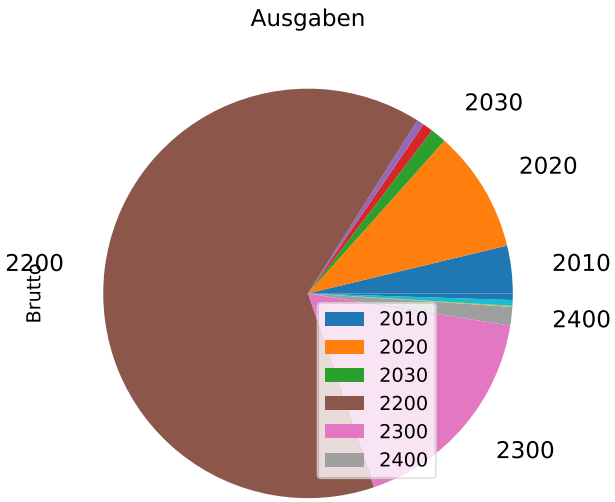
```
ax = ausgaben.plot(y='Brutto',
                   title='Ausgaben',
                   kind="pie",
                   labels=[''] * len(ausgaben))

ax.legend(bbox_to_anchor=(0.5, 0.5),
          labels=ausgaben.index)
```



Bei der zweiten Möglichkeit unterdrücken wir die Ausgabe, wenn die Werte unterhalb einer bestimmten Schwelle liegen. Dazu kopieren wir das DataFrame und setzen dort die entsprechenden Labels im Index auf einen leeren String. Zuerst wird der DataFrame `ausgaben` verwendet, um die prozentualen Anteile jeder Kategorie zu berechnen. Die Werte werden dann der Variable `ausg_norm` zugewiesen. Anschließend wird der Index von `ausg_norm` basierend auf den Werten in der Spalte `Brutto` geändert, sodass Kategorien mit einem Anteil von weniger als 1 % nicht im Diagramm dargestellt werden. Die Lambda-Funktion wird verwendet, um den Index von `ausg_norm` zu ändern. Wenn der Bruttoanteil einer Kategorie kleiner als 1 ist, wird der Index-Wert auf einen leeren String (”) gesetzt, ansonsten wird der Index-Wert unverändert beibehalten.

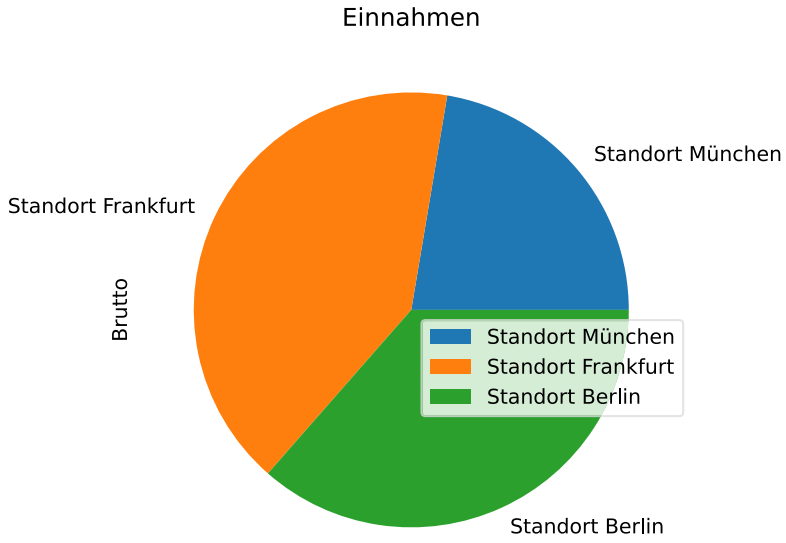
Das kreisförmige Diagramm wird dann mit der `plot`-Methode von Pandas erstellt, indem `kind="pie"` angegeben wird. Die Ausgabenkategorien werden als Labels in den Abschnitten des Diagramms angezeigt. Die `labeldistance`-Option legt die Entfernung der Labels vom Zentrum des Diagramms fest, während die `textprops`-Option verwendet wird, um die Schriftgröße der Labels anzupassen. Eine Legende wird mit der `legend`-Methode von Matplotlib erstellt und auf der linken oberen Seite des Diagramms platziert.

```
ausg_norm = ausgaben * 100 / ausgaben.sum()
print(ausg_norm)
ausg_norm.index = ausg_norm.index.map(lambda x: '' \
                        if ausg_norm.loc[x, 'Brutto'] < 1 else x)

ax = ausg_norm.plot(y='Brutto',
                    title='Ausgaben',
                    labeldistance=1.2,
                    textprops={'fontsize': 12},
                    kind="pie")
ax.legend(bbox_to_anchor=(0.5, 0.5),
          loc="upper left")
```

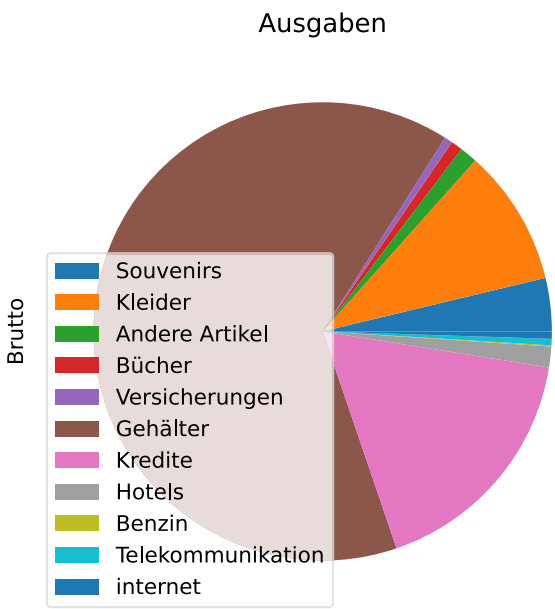


Nun mit den Kontennamen statt den Kontonummern für die Einnahmen:

```
beschreibungen = konto_beschreibung["Beschreibung"].loc[einnahmen.index]

ax = einnahmen.plot(y='Brutto',
                    title='Einnahmen',
                    labels=beschreibungen,
                    kind="pie")
ax.legend(bbox_to_anchor=(0.5, 0.5),
          labels=beschreibungen,
          loc="upper left")
```

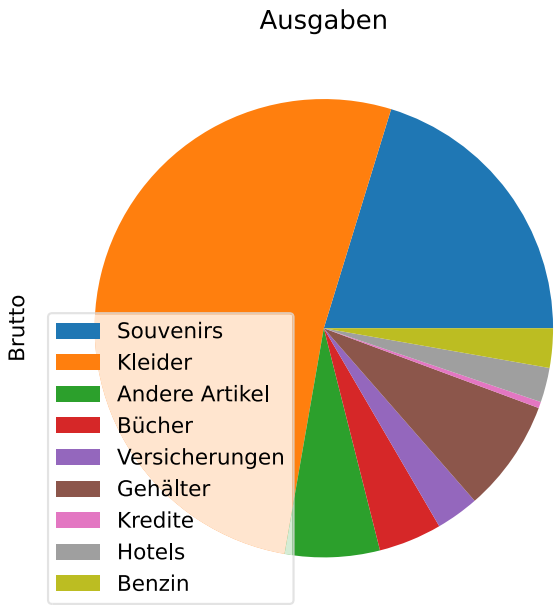



```
beschreibungen = konto_beschreibung["Beschreibung"].loc[ausgaben.index]

ax = ausgaben.plot(y='Brutto',
                   title='Ausgaben',
                   kind="pie",
                   labels=[''] * len(ausgaben))

ax.legend(loc="lower left",
          labels=beschreibungen)
```



Die Ausgabenkonten mit den niedrigen Salden sind nur sehr schlecht in unseren Plots erkennbar. Eine Möglichkeit, dies zu verbessern, besteht darin, die großen Posten wie Gehälter (2200) und Kredite (2300) in der Ausgabe zu unterdrücken:

```
beschreibungen = konto_beschreibung["Beschreibung"].loc[ausgaben.index]

ax = ausgaben.drop([2200, 2300]).plot(y='Brutto',
                                      title='Ausgaben',
                                      kind="pie",
                                      labels=[''] * len(ausgaben))

ax.legend(loc="lower left",
          labels=beschreibungen)
```



31.2.3 Steuersummen

Wir wollen nun die Steuersummen berechnen. Wir benutzen dazu die Spalte `StSatz`, die den Steuersatz enthält. Im Folgenden definieren wir nun eine Funktion `steuersummen`, die die Mehrwertsteuersummen nach Steuersätzen aus einem Journal-DataFrame berechnet:

```
import pandas as pd


def steuersummen(journal_df, monate=None):
    """ Liefert ein DataFrame mit den Umsätzen und Steuersätzen
    zurück. Wird monate eine Zahl oder Liste übergeben, werden
    nur die Umsätze der entsprechenden Monate berücksichtigt.
    Beispiel: steuersummen(df, monate=[3, 6]) bedeutet nur die
    Monate 3 (März) und 6 (Juni)"""
```

```
    if monate:
        if isinstance(monate, int):
            monate_cond = journal_df.index.month == monate
        elif isinstance(monate, (list, tuple)):
            monate_cond = journal_df.index.month.isin(monate)
        positive = journal_df["Brutto"] > 0
        umsatzsteuern = journal_df[positive & monate_cond]
        negative = journal_df["Brutto"] < 0
        vorsteuern = journal_df[negative & monate_cond]
    else:
        umsatzsteuern = journal_df[journal_df["Brutto"] > 0]
        vorsteuern = journal_df[journal_df["Brutto"] < 0]

    umsatzsteuern = umsatzsteuern[["StSatz", "Brutto"]].groupby("StSatz").sum()
    umsatzsteuern.rename(columns={"Brutto": "Umsaetze brutto"},
                         inplace=True)
    umsatzsteuern.index.name = 'Steuerrate'

    vorsteuern = vorsteuern[["StSatz", "Brutto"]].groupby("StSatz").sum()
    vorsteuern.rename(columns={"Brutto": "Ausgaben brutto"},
                      inplace=True)
    vorsteuern.index.name = 'Steuerrate'

    steuern = pd.concat([vorsteuern, umsatzsteuern], axis=1)
    steuern.insert(1,
                   column="Vorsteuer",
                   value=(steuern["Ausgaben brutto"] * steuern.index /
                   ↪  100).round(2))
    steuern.insert(3,
                   column="Umsatzsteuer",
                   value=(steuern["Umsaetze brutto"] * steuern.index /
                   ↪  100).round(2))

    return steuern.fillna(0)
```

Wir lesen die Daten wieder aus unserer Excel-Datei ein und berechnen dann die Steuersummen nach Steuersätzen mit der Funktion `steuersummen` und `journal` als Argument:

```
with pd.ExcelFile("einnahme_ueber_2022.xlsx") as xl:
    konto_beschreibung = xl.parse("kontenplan",
                                  index_col=0)
    journal = xl.parse("journal",
                       index_col=0,
                       )
sts = steuersummen(journal)
print(sts)
```

Ausgabe:

```
             Ausgaben brutto  Vorsteuer  Umsaetze brutto  Umsatzsteuer
Steuerrate
0                  -90102.20      -0.00          8334.43          0.00
7                   -3847.10    -269.30         11240.71        786.85
19                 -14948.32   -2840.18        149401.04      28386.20
```

Die Steuersummen für den Monat Mai berechnet man wie folgt:

```
sts = steuersummen(journal, monate=5)
print(sts)
```

Ausgabe:

```
             Ausgaben brutto  Vorsteuer  Umsaetze brutto  Umsatzsteuer
Steuerrate
0                  -22411.53      -0.00             0.00          0.00
7                    -900.00     -63.00             0.00          0.00
19                   -145.00     -27.55         31328.98       5952.51
```

Jetzt berechnen wir die Steuern für März und April:

```
sts = steuersummen(journal, monate=[3, 4])
print(sts)
```

Ausgabe:

```
             Ausgaben brutto  Vorsteuer  Umsaetze brutto  Umsatzsteuer
Steuerrate
0                  -22878.53      -0.00          1854.96          0.00
7                   -1239.10     -86.74          6259.87        438.19
19                  -8480.20   -1611.24         37061.01       7041.59
```